Part 4

Architecting on AWS

Werner Vogels, CTO of Amazon.com, is quoted as saying "Everything fails all the time." Instead of trying to reach the unreachable goal of an unbreakable system, AWS plans for failure:

- Hard drives can fail, so S3 stores data on multiple hard drives to prevent loss of data.
- Computing hardware can fail, so virtual machines can be automatically restarted on another machine if necessary.
- Data centers can fail, so there are multiple data centers per region that can be used in parallel or on demand.

Outages of IT infrastructure and applications can cause loss of trust and money, and are a major risk for business. You will learn how to prevent an outage of your AWS applications by using the right tools and architecture.

Some AWS services handle failure by default in the background. For some services, responding to failure scenarios is available on demand. And some services don't handle failure by themselves, but offer the possibility to plan and react to failure. The following table shows an overview of the most important services and their failure handling.

Designing for failure is a fundamental principle on AWS. Another one is to make use of the elasticity of the cloud. You will learn about how to increase the number of your virtual machines based on the current workload. This will allow you to architect reliable systems for AWS.

Overview of services and their failure handling possibilities

	Description	Examples
Fault tolerant	Services can recover from failure automatically without any downtime.	S3 (object storage), DynamoDB (NoSQL database), Route 53 (DNS)
Highly available	Services can recover from some failures with a small downtime automatically.	RDS (relational database), EBS (network attached storage)
Manual failure handling	Services do not recover from failure by default but offer tools to build a highly available infrastructure on top of them.	EC2 (virtual machine)

Chapter 14 lays the foundation for becoming independent of the risk of losing a single server or a complete data center. You will learn how to recover a single EC2 instance either in the same data center or in another data center.

Chapter 15 introduces the concept of decoupling your system to increase reliability. You will learn how to use synchronous decoupling with the help of load balancers on AWS. You'll also see asynchronous decoupling by using Amazon SQS, a distributed queuing service, to build a fault-tolerant system.

Chapter 16 uses a lot of the services you've discovered so far to built a fault-tolerant application. You'll learn everything you need to design a fault-tolerant web application based on EC2 instances (which aren't fault-tolerant by default).

Chapter 17 is all about elasticity. You will learn how to scale your capacity based on a schedule, or based on the current load of your system.

14 Achieving high availability: availability zones, auto-scaling, and CloudWatch

This chapter covers

- Using a CloudWatch alarm to recover a failed virtual machine
- Understanding availability zones in an AWS region
- Using auto-scaling to guarantee your VMs keep running
- Analyzing disaster-recovery requirements

Imagine you run a web shop. During the night, the hardware running your virtual machine fails. Until the next morning when you go into work, your users can no longer access your web shop. During the 8-hour downtime, your users search for an alternative and stop buying from you. That's a disaster for any business. Now imagine a highly available web shop. Just a few minutes after the hardware failed, the system recovers, restarts itself on new hardware, and your web shop is back online again—

without any human intervention. Your users can now continue to shop on your site. In this chapter, we'll teach you how to build a high-availability architecture based on EC2 instances.

Virtual machines aren't highly available by default. The following scenarios could cause an outage of your virtual machine:

- A software issue causes the virtual machine's OS to fail.
- A software issue occurs on the host machine, causing the VM to crash (either the OS of the host machine crashes or the virtualization layer does).
- The computing, storage, or networking hardware of the physical host fails.
- Parts of the data center that the virtual machine depends on fail: network connectivity, the power supply, or the cooling system.

For example, if the computing hardware of a physical host fails, all EC2 instances running on this host will fail. If you're running an application on an affected virtual machine, this application will fail and experience downtime until somebody—probably you—intervenes by starting a new virtual machine on another physical host. To avoid downtimes, you should enable auto recovery or use multiple virtual machines.

Examples are 100% covered by the Free Tier

The examples in this chapter are totally covered by the Free Tier. As long as you don't run the examples longer than a few days, you won't pay anything for it. Keep in mind that this applies only if you created a fresh AWS account for this book and there is nothing else going on in your AWS account. Try to complete the chapter within a few days, because you'll clean up your account at the end of the chapter.

High availability describes a system that is operating with almost no downtime. Even if a failure occurs, the system can provide its services most of the time (for example, 99.99% over a year). Although a short interruption might be necessary to recover from a failure, there is no need for human interaction. The Harvard Research Group (HRG) defines high availability with the classification AEC-2, which requires an uptime of 99.99% over a year, or not more than 52 minutes and 35.7 seconds of downtime per year. You can achieve 99.99% uptime with EC2 instances if you follow the instructions in the rest of this chapter.

High availability vs. fault tolerance

A highly available system can recover from a failure automatically with a short downtime. A fault-tolerant system, in contrast, requires the system to provide its services without interruption in case of a component failure. We'll show you how to build a fault-tolerant system in chapter 16.

AWS offers tools for building highly available systems based on EC2 instances:

- Monitoring the health of virtual machines with CloudWatch and triggering recovery automatically if needed
- Building a highly available infrastructure by using groups of isolated data centers, called availability zones, within a region
- Using auto-scaling to guarantee a certain number of virtual machines will keep running, and replace failed instances automatically

14.1 Recovering from EC2 instance failure with CloudWatch

The EC2 service checks the status of every virtual machine automatically. System status checks are performed every minute and the results are available as CloudWatch metrics.

> **AWS CloudWatch**
>
> *AWS CloudWatch* is a service offering metrics, events, logs, and alarms for AWS resources. You used CloudWatch to monitor a Lambda function in chapter 7, and gained some insight into the current load of a relational database instance in chapter 11.

A *system status check* detects a loss of network connectivity or power, as well as software or hardware issues on the physical host. AWS then needs to repair failures detected by the system status check. One possible strategy to resolve such failures is to move the virtual machine to another physical host.

Figure 14.1 shows the process in the case of an outage affecting a virtual machine:

1. The physical hardware fails and causes the EC2 instance to fail as well.
2. The EC2 service detects the outage and reports the failure to a CloudWatch metric.
3. A CloudWatch alarm triggers recovery of the virtual machine.
4. The EC2 instance is launched on another physical host.
5. The EBS volume and Elastic IP stay the same, and are linked to the new EC2 instance.

After the recovery, a new EC2 instance is running with the same ID and private IP address. Data on network-attached EBS volumes is available as well. No data is lost because the EBS volume stays the same. EC2 instances with local disks (instance storage) aren't supported for this process. If the old EC2 instance was connected to an Elastic IP address, the new EC2 instance is connected to the same Elastic IP address.

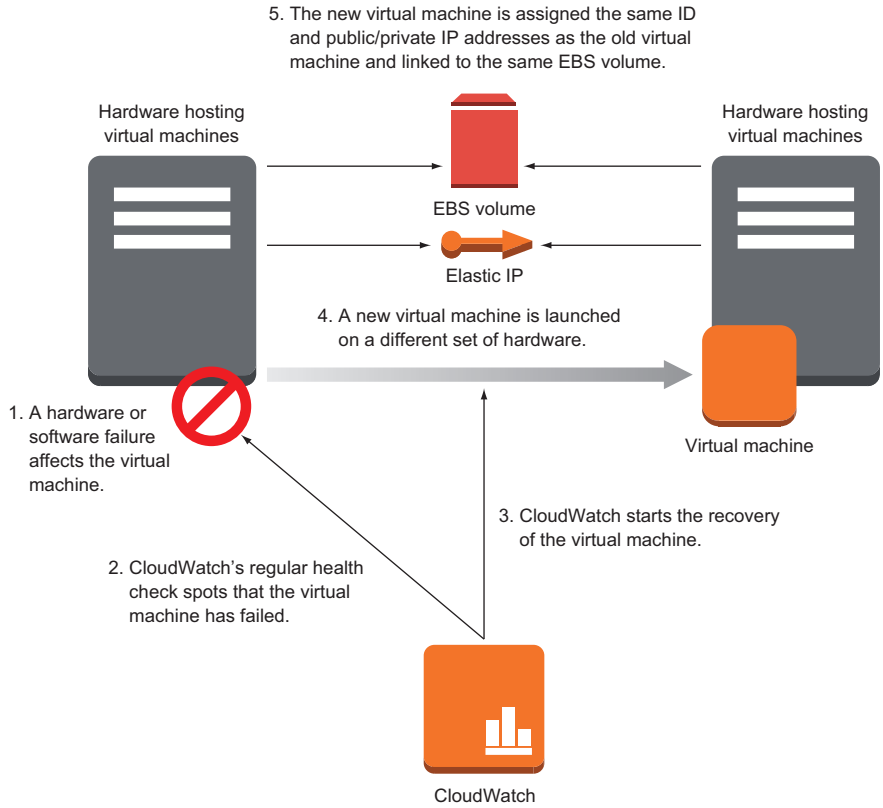


Figure 14.1 In the case of a hardware failure, CloudWatch triggers the recovery of the EC2 instance.

Requirements for recovering EC2 instances

An EC2 instance must meet the following requirements if you want to use the recovery feature:

- It must be running in a VPC network.
- The instance family must be C3, C4, C5, M3, M4, M5, R3, R4, T2, or X1. Other instance families aren't supported.
- The EC2 instance must use EBS volumes exclusively, because data on instance storage would be lost after the instance was recovered.

14.1.1 Creating a CloudWatch alarm to trigger recovery when status checks fail

A CloudWatch alarm consists of the following:

- A metric that monitors data (health check, CPU usage, and so on)
- A rule defining a threshold based on a statistical function over a period of time
- Actions to trigger if the state of the alarm changes (such as triggering a recovery of an EC2 instance if the state changes to `ALARM`)

The following states are available for an alarm:

- `OK`—Everything is fine; the threshold hasn't been reached.
- `INSUFFICIENT_DATA`—There isn't enough data to evaluate the alarm.
- `ALARM`—Something is broken: the threshold has been overstepped.

To monitor a VM's health and recover it in case the underlying host system fails, you can use a CloudWatch alarm like the one shown in listing 14.1. This listing is an excerpt from a CloudFormation template.

Listing 14.1 creates a CloudWatch alarm based on a metric called `StatusCheckFailed_System` (linked by attribute `MetricName`). This metric contains the results of the system status checks performed by the EC2 service every minute. If the check fails, a measurement point with value `1` is added to the metric `StatusCheckFailed_System`. Because the EC2 service publishes this metric, the `Namespace` is called `AWS/EC2` and the `Dimension` of the metric is the ID of a virtual machine.

The CloudWatch alarm checks the metric every 60 seconds, as defined by the `Period` attribute. As defined in `EvaluationPeriods`, the alarm will check the last five periods, the last 5 minutes in this example. The check runs a statistical function specified in `Statistic` on the time periods. The result of the statistical function, a minimum function in this case, is compared against `Threshold` using the chosen `ComparisonOperator`. If the result is negative, the alarm actions defined in `AlarmActions` are executed: in this case, the recovery of the virtual machine—a built-in action for EC2 instances.

Listing 14.1 Creating a CloudWatch alarm to monitor the health of an EC2 instance

```
# [...]
RecoveryAlarm:
  Type: 'AWS::CloudWatch::Alarm'
  Properties:
    AlarmDescription: 'Recover EC2 instance ...'
    Namespace: 'AWS/EC2'
    MetricName: 'StatusCheckFailed_System'
    Statistic: Maximum
    Period: 60
    EvaluationPeriods: 5
```

Creates a CloudWatch alarm to monitor the VM's health

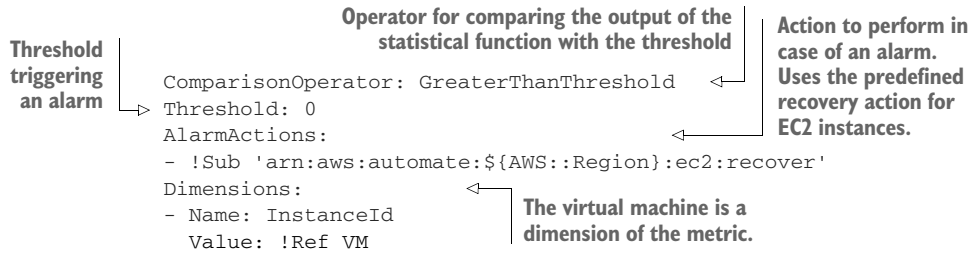
The metric to monitor is provided by the EC2 service with namespace AWS/EC2.

The name of the metric.

Statistical function to apply to the metric.

Number of time periods over which data is compared to the threshold

Duration the statistical function is applied, in seconds. Must be a multiple of 60.



In summary, AWS checks the status of the virtual machine every minute. The result of these checks is written to the `StatusCheckFailed_System` metric. The alarm checks this metric. If there are five consecutive failed checks, the alarm trips.

14.1.2 Monitoring and recovering a virtual machine based on a CloudWatch alarm

Suppose that your team is using an agile development process. To accelerate the process, your team decides to automate the testing, building, and deployment of the software. You've been asked to set up a continuous integration (CI) server. You've chosen to use Jenkins, an open source application written in Java that runs in a servlet container such as Apache Tomcat. Because you're using infrastructure as code, you're planning to deploy changes to your infrastructure with Jenkins as well.[1]

A Jenkins server is a typical use case for a high-availability setup. It's an important part of your infrastructure, because your colleagues won't be able to test and deploy new software if Jenkins suffers from downtime. But a short downtime in the case of a failure with automatic recovery won't hurt your business too much, so you don't need a fault-tolerant system. Jenkins is only an example. You can apply the same principles to any other applications where you can tolerate a short amount of downtime but still want to recover from hardware failures automatically. For example, we used the same approach for hosting FTP servers and VPN servers.

In this example, you'll do the following:

1. Create a virtual network in the cloud (VPC).
2. Launch a virtual machine in the VPC, and automatically install Jenkins during bootstrap.
3. Create a CloudWatch alarm to monitor the health of the virtual machine.

We'll guide you through these steps with the help of a CloudFormation template.

You can find the CloudFormation template for this example on GitHub and on S3. You can download a snapshot of the repository at http://mng.bz/x6RP. The file we're talking about is located at chapter14/recovery.yaml. On S3, the same file is located at http://mng.bz/994D.

[1] Learn more about Jenkins by reading its documentation at http://mng.bz/sVqd.

The following command creates a CloudFormation template which launches an EC2 instance with a CloudWatch alarm that triggers a recovery if the virtual machine fails. Replace *$Password* with a password consisting of 8–40 characters. The template automatically installs a Jenkins server while starting the virtual machine:

```
$ aws cloudformation create-stack --stack-name jenkins-recovery \
➥ --template-url https://s3.amazonaws.com/\
➥ awsinaction-code2/chapter14/recovery.yaml \
➥ --parameters ParameterKey=JenkinsAdminPassword,ParameterValue=$Password
```

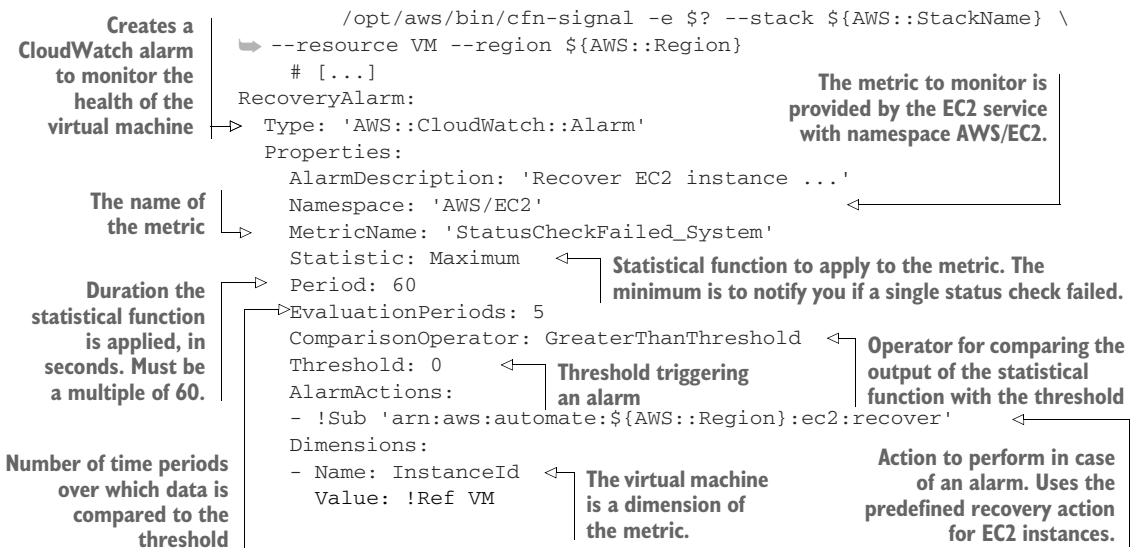
The CloudFormation template contains the definition of a private network and security configuration. But the most important parts of the template are these:

- A virtual machine with user data containing a Bash script, which installs a Jenkins server during bootstrapping
- A public IP address assigned to the EC2 instance, so you can access the new instance after a recovery using the same public IP address as before
- A CloudWatch alarm based on the system-status metric published by the EC2 service

The following listing shows the important parts of the template.

Listing 14.2 Starting an EC2 instance running a Jenkins CI server with a recovery alarm

```
#[...]
ElasticIP:                                  # The public IP address stays the same after recovery when using ElasticIP.
  Type: 'AWS::EC2::EIP'
  Properties:
    InstanceId: !Ref VM
    Domain: vpc
  DependsOn: GatewayToInternet
VM:                                         # Launches a virtual machine to run a Jenkins server
  Type: 'AWS::EC2::Instance'
  Properties:
    ImageId: 'ami-6057e21a'                 # Selects the AMI (in this case Amazon Linux)
    InstanceType: 't2.micro'                # Recovery is supported for t2 instance types.
    KeyName: mykey
    NetworkInterfaces:
    - AssociatePublicIpAddress: true
      DeleteOnTermination: true
      DeviceIndex: 0
      GroupSet:
      - !Ref SecurityGroup
      SubnetId: !Ref Subnet
    UserData:                               # User data containing a shell script that is executed during bootstrapping to install a Jenkins server
      'Fn::Base64': !Sub |
        #!/bin/bash -x
        bash -ex << "TRY"                   # Downloads and installs Jenkins
          wget -q -T 60 https://.../jenkins-1.616-1.1.noarch.rpm
          rpm --install jenkins-1.616-1.1.noarch.rpm
          # configure Jenkins [...]
          service jenkins start             # Starts Jenkins
        TRY
```



It will take a few minutes for the CloudFormation stack to be created and Jenkins to be installed on the virtual machine. Run the following command to get the output of the stack. If the output is null, retry after a few more minutes:

```
$ aws cloudformation describe-stacks --stack-name jenkins-recovery \
➥ --query "Stacks[0].Outputs"
```

If the query returns output like the following, containing a URL, a user, and a password, the stack has been created and the Jenkins server is ready to use. If you want more information during stack creation, we recommend you use the CloudFormation Management Console at https://console.aws.amazon.com/cloudformation/. Open the URL in your browser, and log in to the Jenkins server with user `admin` and the password you've chosen:

```
[
  {
    "Description": "URL to access web interface of Jenkins server.",
    "OutputKey": "JenkinsURL",
    "OutputValue": "http://54.152.240.91:8080"
  },
  {
    "Description": "Administrator user for Jenkins.",
    "OutputKey": "User",
    "OutputValue": "admin"
  },
  {
    "Description": "Password for Jenkins administrator user.",
    "OutputKey": "Password",
    "OutputValue": "********"
  }
]
```

Open this URL in your browser to access the web interface of the Jenkins server.

Use this user name to log in to the Jenkins server.

Use this password to log in to the Jenkins server.

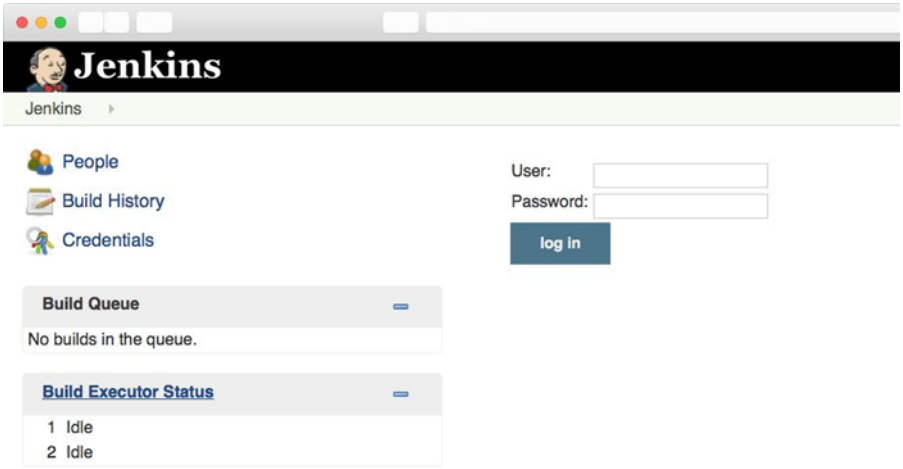


Figure 14.2 Web interface of the Jenkins server

You're now ready to create your first build job on the Jenkins server. To do so, you have to log in with the username and password from the previous output. Figure 14.2 shows the Jenkins server's login form.

Once you're logged in, you can create your first job by following these steps:

1. Click New Item in the navigation bar on the left.
2. Type `AWS in Action` as the name for the new job.
3. Select Freestyle Project as the job type, and click OK to save the job.

The Jenkins server runs on a virtual machine with automated recovery. If the virtual machine fails because of issues with the host system, it will be recovered with all data and the same public IP address. The URL doesn't change because you're using an Elastic IP for the virtual machine. All data is restored because the new virtual machine uses the same EBS volume as the previous virtual machine, so you can find your *AWS in Action* job again.

Unfortunately, you can't test the recovery process. The CloudWatch alarm monitors the health of the host system, which can only be controlled by AWS.

Cleaning up

Now that you've finished this example, it's time to clean up to avoid unwanted charges. Execute the following command to delete all resources corresponding to the Jenkins setup:

```
$ aws cloudformation delete-stack --stack-name jenkins-recovery
$ aws cloudformation wait stack-delete-complete \
➥ --stack-name jenkins-recovery
```

Waits until the stack is deleted

14.2 Recovering from a data center outage

Recovering an EC2 instance after underlying software or hardware fails is possible using system status checks and CloudWatch, as described in the previous section. But what happens if the entire data center fails because of a power outage, a fire, or some other issue? Recovering a virtual machine as described in section 14.1 will fail because it tries to launch an EC2 instance in the same data center.

AWS is built for failure, even in the rare case that an entire data center fails. The AWS regions consist of multiple data centers grouped into availability zones. Auto-scaling helps you start virtual machines that can recover from a data center outage with only a short amount of downtime. There are two pitfalls when building a highly available setup over multiple availability zones:

1. Data stored on network-attached storage (EBS) won't be available after failing over to another availability zone by default. So you can end up having no access to your data (stored on EBS volumes) until the availability zone is back online (you won't lose your data in this case).
2. You can't start a new virtual machine in another availability zone with the same private IP address. As you've learned, subnets are bound to availability zones, and each subnet has a unique IP address range. By default, you can't keep the same public IP address automatically after a recovery, as was the case in the previous section with a CloudWatch alarm triggering a recovery.

In this section, you'll improve the Jenkins setup from the previous section, add the ability to recover from an outage of an entire availability zone, and work around the pitfalls afterward.

14.2.1 Availability zones: groups of isolated data centers

As you've learned, AWS operates multiple locations worldwide, called regions. You've used region US East (N. Virginia), also called us-east-1, if you've followed the examples so far. In total, there are 15 publicly available regions throughout North America, South America, Europe, and Asia Pacific.

Each region consists of multiple availability zones (AZs). You can think of an AZ as an isolated group of data centers, and a region as an area where multiple availability zones are located at a sufficient distance. The region us-east-1 consists of six availability zones (us-east-1a to us-east-1f) for example. The availability zone us-east-1a could be one data center, or many. We don't know because AWS doesn't make information about their data centers publicly available. So from an AWS user's perspective, you only know about regions and AZs.

The AZs are connected through low-latency links, so requests between different availability zones aren't as expensive as requests across the internet in terms of latency. The latency within an availability zone (such as from an EC2 instance to another EC2 instance in the same subnet) is lower compared to latency across AZs. The number of availability zones depends on the region. Most regions come with three or more

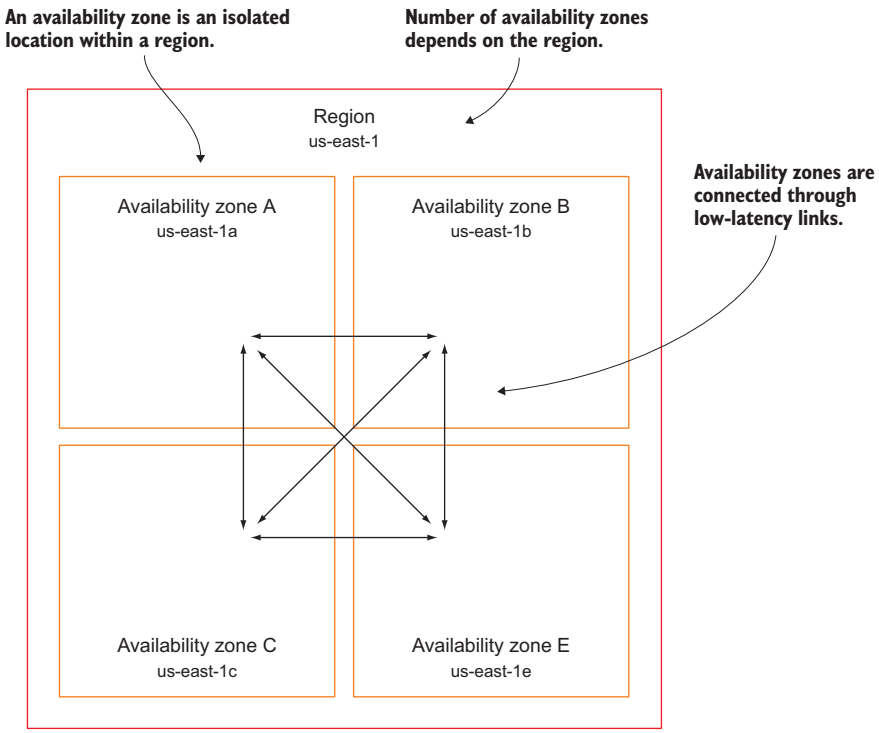


Figure 14.3 A region consists of multiple availability zones connected through low-latency links.

availability zones. When choosing a region, keep in mind that AWS also has regions with only two availability zones. This could become an issue if you want to run a distributed system that relies on consensus decisions. Figure 14.3 illustrates the concept of availability zones within a region.

Some AWS services are highly available or even fault-tolerant by default. Other services provide building blocks to achieve a highly available architecture. You can use multiple availability zones or even multiple regions to build a highly available architecture, as figure 14.4 shows:

- Some services operate globally over multiple regions: Route 53 (DNS) and CloudFront (CDN).
- Some services use multiple availability zones within a region so they can recover from an availability zone outage: S3 (object store) and DynamoDB (NoSQL database).
- The Relational Database Service (RDS) offers the ability to deploy a master-standby setup, called *Multi-AZ deployment*, so you can fail over into another availability zone with a short downtime if necessary.
- A virtual machine runs in a single availability zone. But AWS offers tools to build an architecture based on EC2 instances that can fail over into another availability zone.

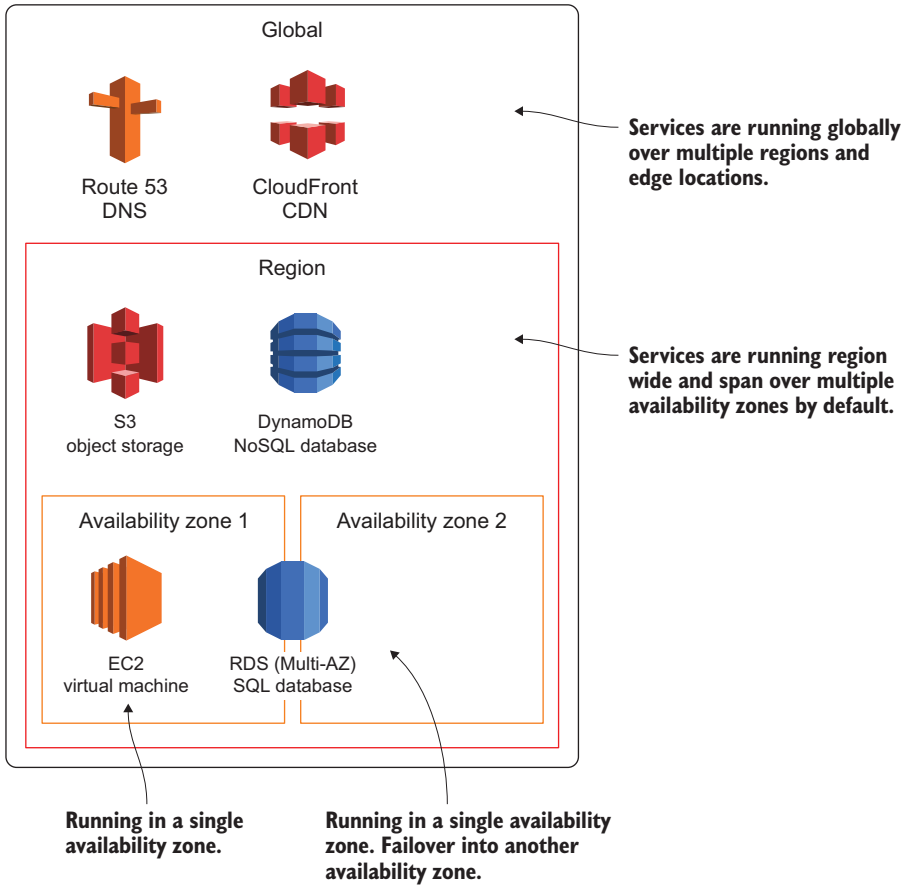


Figure 14.4 AWS services can operate in a single availability zone, over multiple availability zones within a region, or even globally.

The identifier for an availability zone consists of the identifier for the region (such as us-east-1) and a character (a, b, c, ...). So us-east-1a is the identifier for an availability zone in region us-east-1. To distribute resources across the different availability zones, the AZ identifier is generated randomly for each AWS account. This means us-east-1a points to a different availability zone in your AWS account than it does in our AWS account.

You can use the following commands to discover all regions available for your AWS account:

```
$ aws ec2 describe-regions
{
    "Regions": [
        {
            "Endpoint": "ec2.ap-south-1.amazonaws.com",
            "RegionName": "ap-south-1"
```

```
        },
        {
            "Endpoint": "ec2.eu-west-2.amazonaws.com",
            "RegionName": "eu-west-2"
        },
        {
            "Endpoint": "ec2.eu-west-1.amazonaws.com",
            "RegionName": "eu-west-1"
        },
        [...]
        {
            "Endpoint": "ec2.us-west-2.amazonaws.com",
            "RegionName": "us-west-2"
        }
    ]
}
```

To list all availability zones for a region, execute the following command and replace *$Region* with `RegionName` from the previous command:

```
$ aws ec2 describe-availability-zones --region $Region
{
    "AvailabilityZones": [
        {
            "State": "available",
            "ZoneName": "us-east-1a",
            "Messages": [],
            "RegionName": "us-east-1"
        },
        {
            "State": "available",
            "ZoneName": "us-east-1b",
            "Messages": [],
            "RegionName": "us-east-1"
        },
        [...]
        {
            "State": "available",
            "ZoneName": "us-east-1f",
            "Messages": [],
            "RegionName": "us-east-1"
        }
    ]
}
```

Before you start to create a high-availability architecture based on EC2 instances with failover to multiple availability zones, there is one more lesson to learn. If you define a private network in AWS with the help of the VPC service, you need to know the following:

- A VPC is always bound to a region.
- A subnet within a VPC is linked to an availability zone.
- A virtual machine is launched into a single subnet.

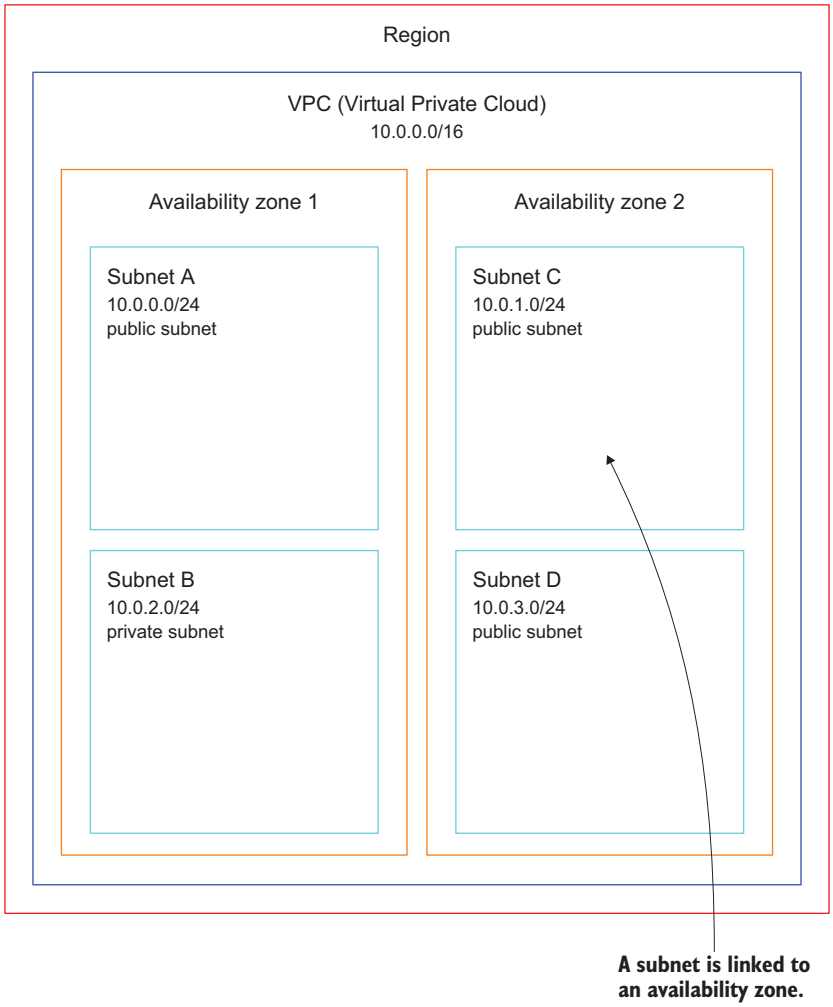


Figure 14.5 A VPC is bound to a region, and a subnet is linked to an availability zone.

Figure 14.5 illustrates these dependencies. Next, you'll learn how to launch a virtual machine that will automatically restart in another availability zone if a failure occurs.

14.2.2 Using auto-scaling to ensure that an EC2 instance is always running

Auto-scaling is part of the EC2 service and helps you to ensure that a specified number of EC2 instances is running even when availability zones become unavailable. You can use auto-scaling to launch a virtual machine and make sure a new instance is started if the original instance fails. You can use it to start virtual machines in multiple subnets. So in case of an outage of an entire availability zone, a new instance can be launched in another subnet in another availability zone.

To configure auto-scaling, you need to create two parts of the configuration:

- A *launch configuration* contains all information needed to launch an EC2 instance: instance type (size of virtual machine) and image (AMI) to start from.
- An *auto-scaling group* tells the EC2 service how many virtual machines should be started with a specific launch configuration, how to monitor the instances, and in which subnets EC2 instances should be started.

Figure 14.6 illustrates this process.

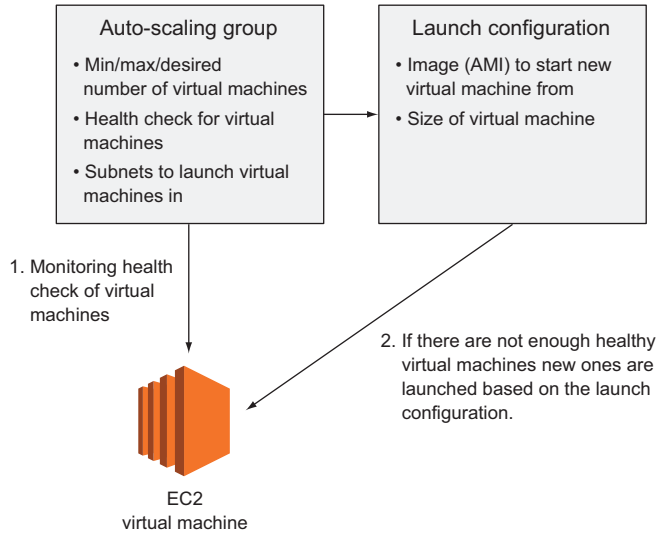


Figure 14.6 Auto-scaling ensures that a specified number of EC2 instances are running.

Listing 14.3 shows how to use auto-scaling to make sure a single EC2 instance is always running. The parameters are explained in table 14.1.

Table 14.1 Required parameters for the launch configuration and auto-scaling group

Context	Property	Description	Values
`LaunchConfiguration`	`ImageId`	The ID of the AMI the virtual machine should be started from.	Any AMI ID accessible from your account.
`LaunchConfiguration`	`InstanceType`	The size of the virtual machine.	All available instance sizes, such as t2.micro, m3.medium, and c3.large.
`AutoScalingGroup`	`DesiredCapacity`	The number of virtual machines that should run in the auto-scaling group at the moment.	Any positive integer. Use `1` if you want a single virtual machine to be started based on the launch configuration.

Table 14.1 Required parameters for the launch configuration and auto-scaling group *(continued)*

Context	Property	Description	Values
`AutoScalingGroup`	`MinSize`	The minimum value for the `DesiredCapacity`	Any positive integer. Use `1` if you want a single virtual machine to be started based on the launch configuration.
`AutoScalingGroup`	`MaxSize`	The maximum value for the `DesiredCapacity`	Any positive integer (greater than or equal to the `MinSize` value). Use 1 if you want a single virtual machine to be started based on the launch configuration.
`AutoScalingGroup`	`VPCZoneIdentifier`	The subnet IDs you want to start virtual machines in	Any subnet ID from a VPC from your account. Subnets must belong to the same VPC.
`AutoScalingGroup`	`HealthCheckType`	The health check used to identify failed virtual machines. If the health check fails, the auto-scaling group replaces the virtual machine with a new one.	`EC2` to use the status checks of the virtual machine, or `ELB` to use the health check of the load balancer (see chapter 16).

An auto-scaling group is also used if you need to scale the number of virtual machines based on usage of your system. You'll learn how to scale the number of EC2 instances based on current load in chapter 17. In this chapter, you only need to make sure a single virtual machine is always running. Because you need a single virtual machine, set the following parameters for auto-scaling to `1`:

- `DesiredCapacity`
- `MinSize`
- `MaxSize`

Listing 14.3 Configuring an auto-scaling group and a launch configuration

```
# [...]
LaunchConfiguration:
  Type: 'AWS::AutoScaling::LaunchConfiguration'
  Properties:
    ImageId: 'ami-6057e21a'
    InstanceType: 't2.micro'
    # [...]
AutoScalingGroup:
  Type: 'AWS::AutoScaling::AutoScalingGroup'
  Properties:
    LaunchConfigurationName: !Ref LaunchConfiguration
```

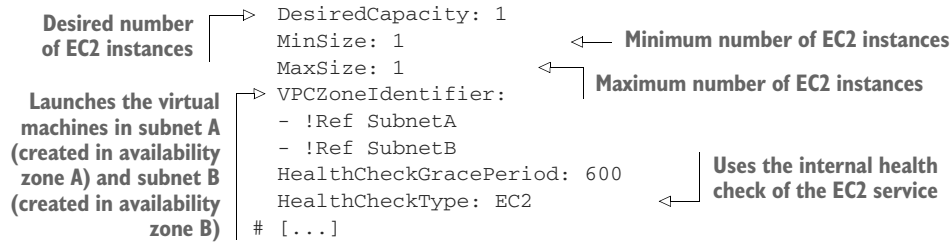
Select the AMI (in this case Amazon Linux).

Launch configuration used for auto-scaling.

Size of the virtual machine

Auto-scaling group responsible for launching the virtual machine

Link to the launch configuration.



The next section will reuse the Jenkins example from the beginning of the chapter to show you how high availability can be achieved with auto-scaling in practice.

14.2.3 Recovering a failed virtual machine to another availability zone with the help of auto-scaling

In the first part of the chapter, you used a CloudWatch alarm to trigger the recovery of a virtual machine that was running a Jenkins CI server, in case of a failure. This mechanism launches an identical copy of the original virtual machine if necessary. This is only possible in the same availability zone, because the private IP address and the EBS volume of a virtual machine are bound to a single subnet and a single availability zone. But suppose your team isn't happy about the fact that they won't be able to use the Jenkins server to test, build, and deploy new software in case of a unlikely availability zone outage. You begin looking for a tool that will let you recover in another availability zone.

Failing over into another availability zone is possible with the help of auto-scaling. You can find the CloudFormation template for this example on GitHub and on S3. You can download a snapshot of the repository at http://mng.bz/x6RP. The file we're talking about is located at chapter14/multiaz.yaml. On S3, the same file is located at http://mng.bz/994D.

Execute the following command to create a virtual machine that can recover in another availability zone if necessary. Replace *$Password* with a password consisting of 8–40 characters. The command uses the CloudFormation template shown in listing 14.4 to set up the environment.

```
$ aws cloudformation create-stack --stack-name jenkins-multiaz \
➥ --template-url https://s3.amazonaws.com/\
➥ awsinaction-code2/chapter14/multiaz.yaml \
➥ --parameters ParameterKey=JenkinsAdminPassword,ParameterValue=$Password
```

You'll find both a launch configuration and an auto-scaling group in the CloudFormation template shown in listing 14.4. You already used the most important parameters for the launch configuration when starting a single virtual machine with a CloudWatch recovery alarm in the previous section:

- `ImageId`—ID of the image (AMI) for virtual machine
- `InstanceType`—Size of the virtual machine
- `KeyName`—Name of the SSH key pair
- `SecurityGroupIds`—Link to the security groups
- `UserData`—Script executed during bootstrap to install the Jenkins CI server

There is one important difference between the definition of a single EC2 instance and the launch configuration: the subnet for the virtual machine isn't defined in the launch configuration, but rather in the auto-scaling group.

Listing 14.4 Launching a Jenkins VM with auto-scaling in two AZs

```
# [...]
LaunchConfiguration:
  Type: 'AWS::AutoScaling::LaunchConfiguration'
  Properties:
    InstanceMonitoring: false
    ImageId: 'ami-6057e21a'
    KeyName: mykey
    SecurityGroups:
    - !Ref SecurityGroup
    AssociatePublicIpAddress: true
    InstanceType: 't2.micro'
    UserData:
      'Fn::Base64': !Sub |
        #!/bin/bash -x
        bash -ex << "TRY"
          wget -q -T 60 https://.../jenkins-1.616-1.1.noarch.rpm
          rpm --install jenkins-1.616-1.1.noarch.rpm
          # [...]
          service jenkins start
        TRY
        /opt/aws/bin/cfn-signal -e $? --stack ${AWS::StackName} \
➥ --resource AutoScalingGroup --region ${AWS::Region}
AutoScalingGroup:
  Type: 'AWS::AutoScaling::AutoScalingGroup'
  Properties:
    LaunchConfigurationName: !Ref LaunchConfiguration
    Tags:
    - Key: Name
      Value: 'jenkins-multiaz'
      PropagateAtLaunch: true
    DesiredCapacity: 1
    MinSize: 1
    MaxSize: 1
    VPCZoneIdentifier:
    - !Ref SubnetA
    - !Ref SubnetB
    HealthCheckGracePeriod: 600
    HealthCheckType: EC2
  CreationPolicy:
    ResourceSignal:
      Timeout: PT10M
```

Launch configuration used for auto-scaling.

Select the AMI (in this case Amazon Linux).

By default, EC2 sends metrics to CloudWatch every 5 minutes. But you can enable detailed instance monitoring to get metrics every minute for an additional cost.

Key for the SSH connections to the virtual machine

Enables the public IP address for the virtual machine

Security groups attached to the virtual machine

Size of the virtual machine

User data containing a script executed during bootstrapping that installs a Jenkins server on the virtual machine

Installs Jenkins

Link to the launch configuration

Auto-scaling group responsible for launching the virtual machine

Tags for the auto-scaling group

Attaches the same tags to the virtual machine started by this auto-scaling group

Desired number of EC2 instances

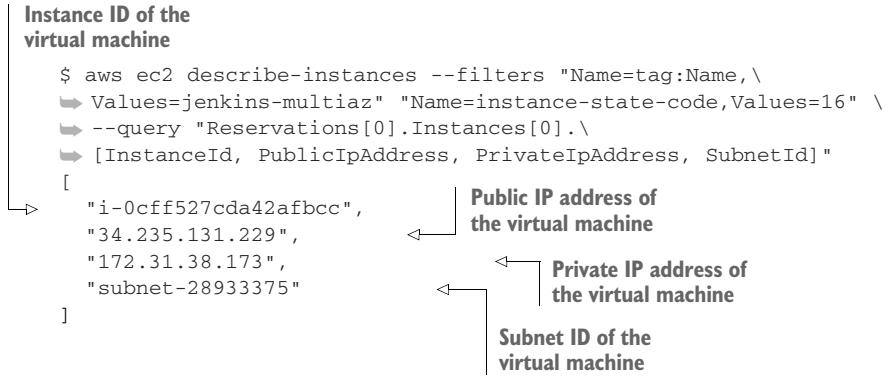
Minimum number of EC2 instances

Maximum number of EC2

Launches the virtual machines in subnet A (created in availability zone A) and subnet B (created in availability zone B)

Uses the internal health check of the EC2 service to discover issues with the virtual machine

The creation of the CloudFormation stack will take a few minutes—time to grab some coffee or tea and take a short break. Execute the following command to grab the public IP address of the virtual machine. If no IP address appears, the virtual machine isn't started yet. Wait another minute, and try again:




```
$ aws ec2 describe-instances --filters "Name=tag:Name,\
➥ Values=jenkins-multiaz" "Name=instance-state-code,Values=16" \
➥ --query "Reservations[0].Instances[0].\
➥ [InstanceId, PublicIpAddress, PrivateIpAddress, SubnetId]"
[
  "i-0cff527cda42afbcc",
  "34.235.131.229",
  "172.31.38.173",
  "subnet-28933375"
]
```

Open http://*$PublicIP*:8080 in your browser, and replace `$PublicIP` with the public IP address from the output of the previous `describe-instances` command. The web interface for the Jenkins server appears.

Execute the following command to terminate the virtual machine and test the recovery process with auto-scaling. Replace `$InstanceId` with the instance ID from the output of the previous `describe` command:

```
$ aws ec2 terminate-instances --instance-ids $InstanceId
```

After a few minutes, the auto-scaling group detects that the virtual machine was terminated and starts a new virtual machine. Rerun the `describe-instances` command until the output contains a new running virtual machine:

```
$ aws ec2 describe-instances --filters "Name=tag:Name,\
➥ Values=jenkins-multiaz" "Name=instance-state-code,Values=16" \
➥ --query "Reservations[0].Instances[0].\
➥ [InstanceId, PublicIpAddress, PrivateIpAddress, SubnetId]"
[
  "i-0293522fad287bdd4",
  "52.3.222.162",
  "172.31.37.78",
  "subnet-45b8c921"
]
```

The instance ID, the public IP address, the private IP address, and probably even the subnet ID have changed for the new instance. Open http://*$PublicIP*:8080 in your browser, and replace `$PublicIP` with the public IP address from the output of the previous `describe-instances` command. The web interface from the Jenkins server appears.

You've now built a highly available architecture consisting of an EC2 instance with the help of auto-scaling. There are two issues with the current setup:

- *The Jenkins server stores data on disk.* When a new virtual machine is started to recover from a failure, this data is lost because a new disk is created.
- *The public and private IP addresses of the Jenkins server change after a new virtual machine is started for recovery.* The Jenkins server is no longer available under the same endpoint.

You'll learn how to solve these problems in the next part of the chapter.

Cleaning up

It's time to clean up to avoid unwanted costs. Execute the following command to delete all resources corresponding to the Jenkins setup:

```
$ aws cloudformation delete-stack --stack-name jenkins-multiaz
$ aws cloudformation wait stack-delete-complete \
➥ --stack-name jenkins-multiaz
```

Waits until the stack is deleted

14.2.4 Pitfall: recovering network-attached storage

The EBS service offers network-attached storage for virtual machines. Remember that EC2 instances are linked to a subnet, and the subnet is linked to an availability zone. EBS volumes are also located in a single availability zone only. If your virtual machine is started in another availability zone because of an outage, the EBS volume cannot be accessed from the other availability zone. Let's say your Jenkins data is stored on an EBS volume in availability zone us-east-1a. As long as you have an EC2 instance running in the same availability zone, you can attach the EBS volume. But if this availability zone becomes unavailable, and you start a new EC2 instance in availability zone us-east-1b, you can't access that EBS volume in us-east-1a, which means that you can't recover Jenkins because you don't have access to the data. See figure 14.7.

Don't mix availability and durability guarantees

An EBS volume is guaranteed to be available for 99.999% of the time. So in case of an availability zone outage, the volume is no longer available. This does not imply that you lose any data. As soon as the availability zone is back online, you can access the EBS volume again with all its data.

An EBS volume guarantees that you won't lose any data in 99.9% of the time. This guarantee is called the durability of the EBS volume. If you have 1,000 volumes in use, you can expect that you will lose one of the volumes and its data a year.

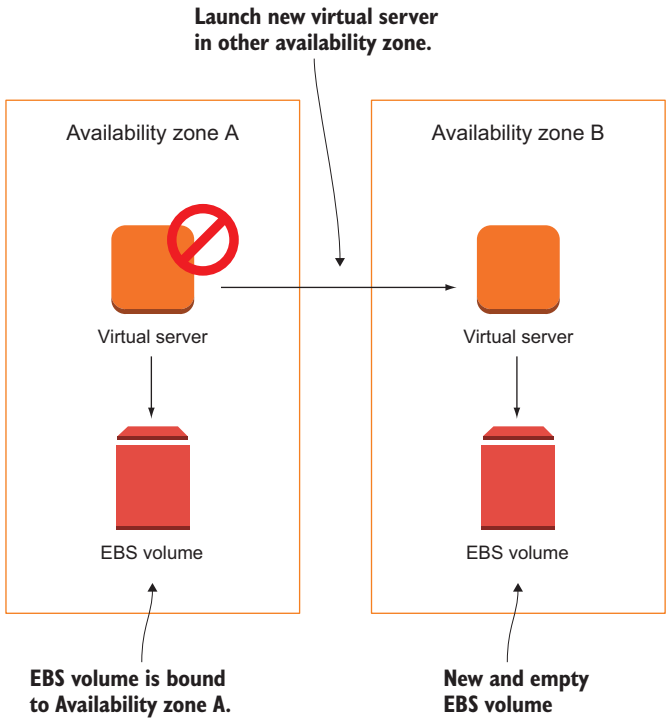


Figure 14.7 An EBS volume is only available in a single availability zone.

There are multiple solutions for this problem:

1. Outsource the state of your virtual machine to a managed service that uses multiple availability zones by default: RDS, DynamoDB (NoSQL database), EFS (NFSv4.1 share), or S3 (object store).
2. Create snapshots of your EBS volumes regularly, and use these snapshots if an EC2 instance needs to recover in another availability zone. EBS snapshots are stored on S3, thus available in multiple availability zones. If the EBS volume is the root volume of the ECS instance, create AMIs to back up the EBS volume instead of a snapshot.
3. Use a distributed third-party storage solution to store your data in multiple availability zones: GlusterFS, DRBD, MongoDB, and so on.

The Jenkins server stores data directly on disk. To outsource the state of the virtual machine, you can't use RDS, DynamoDB, or S3; you need a block-level storage solution instead. As you've learned, an EBS volume is only available in a single availability zone, so this isn't the best fit for the problem. But do you remember EFS from chapter 10? EFS provides block-level storage (over NFSv4.1) and replicates your data automatically between availability zones in a region.

AWS is a fast-growing platform

When we wrote the first edition of this book, EFS was not available. There was basically no easy way to share a filesystem between multiple EC2 instances. As you can imagine, many customers complained about this to AWS. Amazon is proud to be customer obsessed, which means that they listen to their customers carefully. If enough customers need a solution, AWS will deliver a solution. That's why you should follow the new features that are released daily. A problem that was hard to solve yesterday might now be solved by AWS natively. The best place to get updates about AWS is the AWS blog at https://aws.amazon.com/blogs/.

To embed EFS into the Jenkins setup, shown in listing 14.5, you have to make three modifications to the Multi-AZ template from the previous section:

1. Create an EFS filesystem.
2. Create EFS mount targets in each availability zone.
3. Adjust the user data to mount the EFS filesystem. Jenkins stores all its data under `/var/lib/jenkins`.

Listing 14.5 Store Jenkins state on EFS

```
# [...]
FileSystem:
  Type: 'AWS::EFS::FileSystem'
  Properties: {}
MountTargetSecurityGroup:
  Type: 'AWS::EC2::SecurityGroup'
  Properties:
    GroupDescription: 'EFS Mount target'
    SecurityGroupIngress:
    - FromPort: 2049
      IpProtocol: tcp
      SourceSecurityGroupId: !Ref SecurityGroup
      ToPort: 2049
    VpcId: !Ref VPC
MountTargetA:
  Type: 'AWS::EFS::MountTarget'
  Properties:
    FileSystemId: !Ref FileSystem
    SecurityGroups:
    - !Ref MountTargetSecurityGroup
    SubnetId: !Ref SubnetA
MountTargetB:
  Type: 'AWS::EFS::MountTarget'
  Properties:
    FileSystemId: !Ref FileSystem
    SecurityGroups:
    - !Ref MountTargetSecurityGroup
    SubnetId: !Ref SubnetB
# [...]
```

Create the EFS filesystem.

The file system is protected by a security group.

Allows traffic only from the Jenkins EC2 Instances

A mount target is created in subnet A.

Mount target in subnet B

```
LaunchConfiguration:
Type: 'AWS::AutoScaling::LaunchConfiguration'
Properties:
  # [...]
  UserData:
    'Fn::Base64': !Sub |
      #!/bin/bash -x
      bash -ex << "TRY"
        wget -q -T 60 https://.../jenkins-1.616-1.1.noarch.rpm
        rpm --install jenkins-1.616-1.1.noarch.rpm
        while ! nc -z \
➥ ${FileSystem}.efs.${AWS::Region}.amazonaws.com 2049; \
➥ do sleep 10; done
        sleep 10
        echo -n "${FileSystem}.efs.${AWS::Region}.amazonaws.com:/ \
➥ /var/lib/jenkins" >> /etc/fstab
        echo " nfs4 nfsvers=4.1,rsize=1048576,wsize=1048576,hard, \
➥ timeo=600,retrans=2,_netdev 0 0" >> /etc/fstab
        mount -a
        chown jenkins:jenkins /var/lib/jenkins/
        # [...]
        service jenkins start
      TRY
      /opt/aws/bin/cfn-signal -e $? --stack ${AWS::StackName} \
➥ --resource AutoScalingGroup --region ${AWS::Region}
```

Waits until EFS file system is available

Installs Jenkins

Mounts EFS file system

Execute the following command to create the new Jenkins setup that stores state on EFS. Replace *$Password* with a password consisting of 8–40 characters.

```
$ aws cloudformation create-stack --stack-name jenkins-multiaz-efs \
➥ --template-url https://s3.amazonaws.com/\
➥ awsinaction-code2/chapter14/multiaz-efs.yaml \
➥ --parameters ParameterKey=JenkinsAdminPassword,ParameterValue=$Password
```

The creation of the CloudFormation stack will take a few minutes. Run the following command to get the public IP address of the virtual machine. If no IP address appears, the virtual machine isn't started yet. In this case, please wait another minute, and try again:

Instance ID of the virtual machine

```
$ aws ec2 describe-instances --filters "Name=tag:Name,\
➥ Values=jenkins-multiaz-efs" "Name=instance-state-code,Values=16" \
➥ --query "Reservations[0].Instances[0].\
➥ [InstanceId, PublicIpAddress, PrivateIpAddress, SubnetId]"
[
  "i-0efcd2f01a3e3af1d",
  "34.236.255.218",
  "172.31.37.225",
  "subnet-0997e66d"
]
```

Public IP address of the virtual machine

Private IP address of the virtual machine

Subnet ID of the virtual machine

Open http://$PublicIP:8080 in your browser, and replace *$PublicIP* with the public IP address from the output of the previous `describe-instances` command. The web interface from the Jenkins server appears.

Now, create a new Jenkins job by following these steps:

1. Open http://*$PublicIP:8080*/newJob in your browser, and replace *$PublicIP* with the public IP address from the output of the previous `describe` command.
2. Log in with user `admin` and the password you chose when starting the CloudFormation template.
3. Type in `AWS in Action` as the name for the new job.
4. Select Freestyle Project as the job type, and click OK to save the job.

You've made some changes to the state of Jenkins stored on EFS. Now, terminate the EC2 instance with the following command and you will see that Jenkins recovers from the failure without data loss. Replace *$InstanceId* with the instance ID from the output of the previous `describe` command:

```
$ aws ec2 terminate-instances --instance-ids $InstanceId
```

After a few minutes, the auto-scaling group detects that the virtual machine was terminated and starts a new virtual machine. Rerun the `describe-instances` command until the output contains a new running virtual machine:

```
$ aws ec2 describe-instances --filters "Name=tag:Name,\
➥ Values=jenkins-multiaz-efs" "Name=instance-state-code,Values=16" \
➥ --query "Reservations[0].Instances[0].\
➥ [InstanceId, PublicIpAddress, PrivateIpAddress, SubnetId]"
[
  "i-07ce0865adf50cccf",
  "34.200.225.247",
  "172.31.37.199",
  "subnet-0997e66d"
]
```

The instance ID, the public IP address, the private IP address, and probably even the subnet ID have changed for the new instance. Open http://*$PublicIP*:8080 in your browser, and replace *$PublicIP* with the public IP address from the output of the previous `describe-instances` command. The web interface from the Jenkins server appears and it still contains the `AWS in Action` job you created recently.

You've now built a highly available architecture consisting of an EC2 instance with the help of auto-scaling. State is now stored on EFS and is no longer lost when an EC2 instance is replaced. There is one issue left:

- The public and private IP addresses of the Jenkins server change after a new virtual machine is started for recovery. The Jenkins server is no longer available under the same endpoint.

Cleaning up

It's time to clean up to avoid unwanted costs. Execute the following command to delete all resources corresponding to the Jenkins setup:

```
$ aws cloudformation delete-stack --stack-name jenkins-multiaz-efs
$ aws cloudformation wait stack-delete-complete \
➥ --stack-name jenkins-multiaz-efs
```

Waits until the stack is deleted

You'll learn how to solve the last issue next.

14.2.5 Pitfall: network interface recovery

Recovering a virtual machine using a CloudWatch alarm in the same availability zone, as described at the beginning of this chapter, is easy because the private IP address and the public IP address stay the same automatically. You can use these IP addresses as an endpoint to access the EC2 instance even after a failover.

You can't do this when using auto-scaling to recover from a EC2 instance or availability zone outage. If a virtual machine has to be started in another availability zone, it must be started in another subnet. So it's not possible to use the same private IP address for the new virtual machine, as figure 14.8 shows.

By default, you also can't use an Elastic IP as a public IP address for a virtual machine launched by auto-scaling. But the requirement for a static endpoint to receive requests is common. For the use case of a Jenkins server, developers want to bookmark an IP address or a hostname to reach the web interface. There are different possibilities for providing a static endpoint when using auto-scaling to build high availability for a single virtual machine:

- Allocate an Elastic IP, and associate this public IP address during the bootstrap of the virtual machine.
- Create or update a DNS entry linking to the current public or private IP address of the virtual machine.
- Use an Elastic Load Balancer (ELB) as a static endpoint that forwards requests to the current virtual machine.

To use the second solution, you need to link a domain with the Route 53 (DNS) service; we've chosen to skip this solution because you need a registered domain to implement it. The ELB solution is covered in chapter 15, so we'll skip it in this chapter as well. We'll focus on the first solution: allocating an Elastic IP and associating this public IP address during the virtual machine's bootstrap.

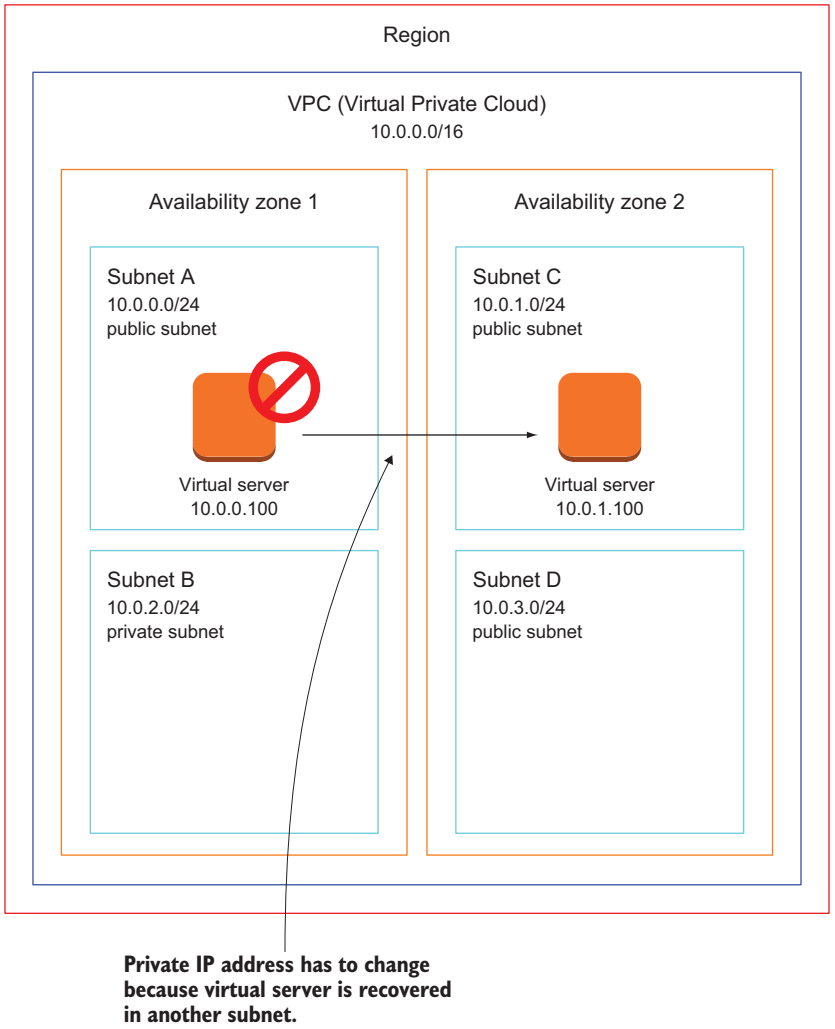


Figure 14.8 The virtual machine starts in another subnet in case of a failover and changes the private IP address.

Execute the following command to create the Jenkins setup based on auto-scaling again, using an Elastic IP address as static endpoint:

```
$ aws cloudformation create-stack --stack-name jenkins-multiaz-efs-eip \
➥ --template-url https://s3.amazonaws.com/\
➥ awsinaction-code2/chapter14/multiaz-efs-eip.yaml \
➥ --parameters ParameterKey=JenkinsAdminPassword,ParameterValue=$Password \
➥ --capabilities CAPABILITY_IAM
```

The command creates a stack based on the template shown in listing 14.6. The differences from the original template spinning up a Jenkins server with auto-scaling are as follows:

- Allocating an Elastic IP
- Adding the association of an Elastic IP to the script in the user data
- Creating an IAM role and policy to allow the EC2 instance to associate an Elastic IP

Listing 14.6 Using an EIP as a static endpoint for a VM launched by auto-scaling

```
# [...]
IamRole:                                    <-- Creates an IAM role used by the EC2 instance
  Type: 'AWS::IAM::Role'
  Properties:
    AssumeRolePolicyDocument:
      Version: '2012-10-17'
      Statement:
      - Effect: Allow
        Principal:
          Service: 'ec2.amazonaws.com'
        Action: 'sts:AssumeRole'
    Policies:
    - PolicyName: root
      PolicyDocument:
        Version: '2012-10-17'
        Statement:
        - Action: 'ec2:AssociateAddress'    <-- Associating an Elastic IP is allowed for EC2 instances using this IAM role.
          Resource: '*'
          Effect: Allow
IamInstanceProfile:
  Type: 'AWS::IAM::InstanceProfile'
  Properties:
    Roles:                                  <-- Allocates an Elastic IP for the virtual machine running Jenkins
    - !Ref IamRole
ElasticIP:
  Type: 'AWS::EC2::EIP'
  Properties:
    Domain: vpc                             <-- Creates an Elastic IP for VPC
LaunchConfiguration:
  Type: 'AWS::AutoScaling::LaunchConfiguration'
  Properties:
    InstanceMonitoring: false
    IamInstanceProfile: !Ref IamInstanceProfile
    ImageId: 'ami-6057e21a'
    KeyName: mykey
    SecurityGroups:
    - !Ref SecurityGroup
    AssociatePublicIpAddress: true
    InstanceType: 't2.micro'
    UserData:
      'Fn::Base64': !Sub |
        #!/bin/bash -x
```

```
      bash -ex << "TRY"
        INSTANCE_ID="$(curl -s http://169.254.169.254/\
➥ latest/meta-data/instance-id)"
        aws --region ${AWS::Region} ec2 associate-address \
➥ --instance-id $INSTANCE_ID \
➥ --allocation-id ${ElasticIP.AllocationId}
        # [...]
        service jenkins start
      TRY
      /opt/aws/bin/cfn-signal -e $? --stack ${AWS::StackName} \
➥ --resource AutoScalingGroup --region ${AWS::Region}
```

Gets the instance ID from the instance metadata

Associates the Elastic IP with the virtual machine

If the query returns output as shown in the following listing, containing a URL, a user, and a password, the stack has been created and the Jenkins server is ready to use. Open the URL in your browser, and log in to the Jenkins server with user `admin` and the password you've chosen. If the output is `null`, try again in a few minutes:

```
$ aws cloudformation describe-stacks --stack-name jenkins-multiaz-efs-eip \
➥ --query "Stacks[0].Outputs"
```

You can now test whether the recovery of the virtual machine works as expected. To do so, you'll need to know the instance ID of the running virtual machine. Run the following command to get this information:

```
$ aws ec2 describe-instances --filters "Name=tag:Name,\
➥ Values=jenkins-multiaz-efs-eip" "Name=instance-state-code,Values=16" \
➥ --query "Reservations[0].Instances[0].InstanceId" --output text
```

Execute the following command to terminate the virtual machine and test the recovery process triggered by auto-scaling. Replace *`$InstanceId`* with the instance from the output of the previous command:

```
$ aws ec2 terminate-instances --instance-ids $InstanceId
```

Wait a few minutes for your virtual machine to recover. Because you're using an Elastic IP assigned to the new virtual machine on bootstrap, you can open the same URL in your browser as you did before the termination of the old instance.

Cleaning up

It's time to clean up to avoid unwanted costs. Execute the following command to delete all resources corresponding to the Jenkins setup:

```
$ aws cloudformation delete-stack --stack-name jenkins-multiaz-efs-eip
$ aws cloudformation wait stack-delete-complete \
➥ --stack-name jenkins-multiaz-efs-eip
```

Waits until the stack is deleted

Now the public IP address of your virtual machine running Jenkins won't change, even if the running virtual machine needs to be replaced by another virtual machine in another availability zone.

14.3 Analyzing disaster-recovery requirements

Before you begin implementing highly available or even fault-tolerant architectures on AWS, you should start by analyzing your disaster-recovery requirements. Disaster recovery is easier and cheaper in the cloud than in a traditional data center, but building for high availability increases the complexity and therefore the initial costs as well as the operating costs of your system. The recovery time objective (RTO) and recovery point objective (RPO) are standards for defining the importance of disaster recovery from a business point of view.

Recovery time objective (RTO) is the time it takes for a system to recover from a failure; it's the length of time until the system reaches a working state again, defined as the system service level, after an outage. In the example with a Jenkins server, the RTO would be the time until a new virtual machine is started and Jenkins is installed and running after a virtual machine or an entire availability zone goes down.

Recovery point objective (RPO) is the acceptable data-loss time caused by a failure. The amount of data loss is measured in time. If an outage happens at 10:00 a.m. and the system recovers with a data snapshot from 09:00 a.m., the time span of the data loss is one hour. In the example of a Jenkins server using auto-scaling, the RPO would be zero, because data is stored on EFS and is not lost during an AZ outage. Figure 14.9 illustrates the definitions of RTO and RPO.

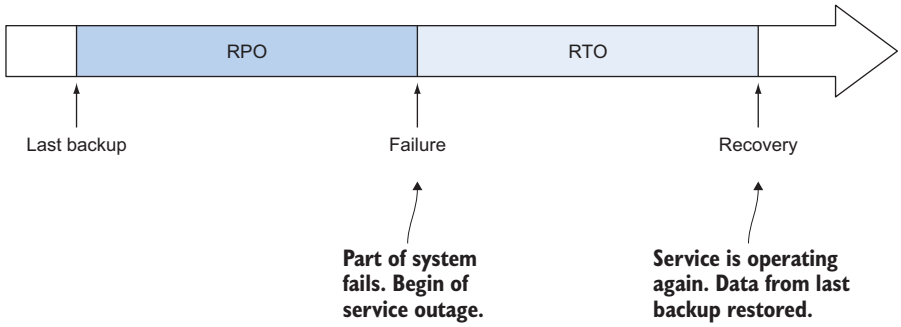


Figure 14.9 Definitions of RTO and RPO

14.3.1 RTO and RPO comparison for a single EC2 instance

You've learned about two possible solutions for making a single EC2 instance highly available. When choosing the solution, you have to know the application's business requirements. Can you tolerate the risk of being unavailable if an availability zone goes down? If so, EC2 instance recovery is the simplest solution where you don't lose any data. If your application needs to survive an unlikely availability zone outage, your safest bet is auto-scaling with data stored on EFS. But this also has performance

impacts compared to storing data on EBS volumes. As you can see, there is no one-size-fits-all solution. You have to pick the solution that fits your business problem best. Table 14.2 compares the solutions.

Table 14.2 Comparison of high availability for a single EC2 instance

	RTO	RPO	Availability
EC2 instance, data stored on EBS root volume: recovery triggered by a CloudWatch alarm	About 10 minutes	No data loss	Recovers from a failure of a virtual machine but not from an outage of an entire availability zone
EC2 instance, data stored on EBS root volume: recovery triggered by auto-scaling	About 10 minutes	All data is lost.	Recovers from a failure of a virtual machine and from an outage of an entire availability zone
EC2 instance, data stored on EBS root volume with regular snapshots: recovery triggered by auto-scaling	About 10 minutes	Realistic time span for snapshots is between 30 minutes and 24 hours.	Recovers from a failure of a virtual machine and from an outage of an entire availability zone
EC2 instance, data stored on EFS filesystem: recovery triggered by auto-scaling	About 10 minutes	No data loss.	Recovers from a failure of a virtual machine and from an outage of an entire availability zone

If you want to be able to recover from an outage of an availability zone and need to decrease the RPO, you should try to achieve a stateless server. Using storage services like RDS, EFS, S3, and DynamoDB can help you to do so. See part 3 if you need help with using these services.

Summary

- A virtual machine fails if the underlying hardware or software fails.
- You can recover a failed virtual machine with the help of a CloudWatch alarm: By default, data stored on EBS, as well as the private and public IP addresses, stays the same.
- An AWS region consists of multiple isolated groups of data centers called availability zones.
- Recovering from an availability zone outage is possible when using multiple availability zones.
- Some AWS services use multiple availability zones by default, but virtual machines run in a single availability zone.
- You can use auto-scaling to guarantee that a single virtual machine is always running even if an availability zone fails. The pitfalls are that you can no longer blindly rely on EBS volumes and by default, IP addresses will change.
- Recovering data in another availability zone is tricky when stored on EBS volumes instead of managed storage services like RDS, EFS, S3, and DynamoDB.